High Performance CPU/GPU Multiresolution Poisson Solver

Wim M. van Rees, Babak Hejazialhosseini, Diego Rossinelli,
Panagiotis Hadjidoukas, Petros Koumoutsakos*
Chair of Computational Science, ETH Zurich, Switzerland

ABSTRACT

We present a multipole-based N-body solver for 3D multiresolution, block-structured grids. The solver is developed for a single heterogeneous CPU/GPU compute node, and evaluates the multipole expansions on the CPU while offloading the particle-particle interactions to the GPU. The block-structure of the destination points is used to guide data parallelism on the CPU, to reduce data transfer to the GPU and to minimize memory accesses during evaluation of the interactions. The algorithmic improvements together with software optimization techniques result in 80% and 97% of the upper bound performance for the CPU and GPU parts, respectively, on a single Cray XK7 compute node.

1. INTRODUCTION

Multipole methods are employed to decrease the computational cost of the N-body problem encountered in many particle-based simulations with applications such as astrophysics and fluid mechanics. Much effort is devoted to optimizing the performance of multipole-based N-body solvers for arbitrarily spaced source and destination particles, both for massively parallel distributed memory architectures [5] and for GPUs and GPU clusters [9].

In the current work we consider incompressible fluid mechanics problems discretized using a remeshed vortex method [4]. Each timestep requires three scalar N-body solutions, once for each of the three vector streamfunction components. Typically in remeshed vortex methods, the regular grid is exploited for FFT-based elliptic solvers [2]. However, the straightforward use of FFT-based solvers is hindered in a multiresolution setting, where the grid spacing changes according to the local spatial scales in the flow. Instead, for multiresolution remeshed vortex methods, multipole methods offer a number of computational advantages. First, they can handle arbitrarily spaced source and destination points. Second, their computational cost scales separately with the number of source and destination points. This is relevant here since the source, the vorticity field, has a compact support, whereas the streamfunction needs to be evaluated globally. Third, they allow for a natural treatment of free-space boundary conditions. And fourth, the granularity of the computational work is relatively fine as every destination point can be evaluated independently, rendering the use of GPU-based acceleration techniques particularly attractive.

Our solver is built on top of MRAG, a wavelet-based, multiresolution block-structured software framework to solve partial differential equations [8]. In MRAG, the computational grid consists of blocks with a fixed number of gridpoints in each direction (here 16). The computational blocks are non-overlapping leaves of a hierarchical tree structure that represents the computational grid, so that each block can exist at a different spatial resolution. Operations across all blocks are parallelized on a multicore node with task-based parallelism, based on the work-stealing principle as implemented in the Intel Threading Building Blocks (TBB) library [6]. More details on MRAG and its parallel implementation on multicore CPUs are given in [7, 8].

2. ALGORITHM

The algorithm used here is based on a Barnes-Hut tree code [1], but adapted for performance as well as for the addressed computational problem. In the preparation phase, we filter all grid points for non-zero vorticity, creating an array of source points spanning only the support of the vorticity field. The source points are hierarchically decomposed into an oct-tree structure, guided by a parameter controlling the maximum number of source points per leaf. For each leaf the multipole expansion coefficients are calculated. At parent nodes the expansions of the children are combined and translated to the parent centers [3].

For the evaluation phase we group a set of destination points into a brick (here 8 grid points in each direction), to increase the amount of work per task. For each destination brick, we create two logical plans containing the lists of all direct and all indirect interactions, respectively. All points in a destination brick are subject to the same logical interaction pattern. Separating the creation of the plan from the actual interaction evaluations enables offloading of all direct interactions to an accelerator such as the GPU. Finally we perform the actual evaluation by executing the logical plans.

*Corresponding author: petros@ethz.ch

3. IMPLEMENTATION

The algorithm is implemented in C++11 and is parallellized using OpenMP tasks and TBB. Throughout the software, instruction irregularities that are known at compile time are resolved using template recursions.

To enable vectorization, the source points are converted from Array of Structures (AoS) to Structure of Arrays (SoA) format, and sorted in Morton order to increase locality. The tree is recursively constructed using OpenMP tasks. In the initial top-down pass we identify and create the nodes on each level, and in a following bottom-up pass we compute the multipole coefficients, vectorized with SSE intrinsics. Subsequently, the logical plan for the evaluation phase is constructed with parallel tasks across the destination bricks.

The evaluation phase is the most computationally expensive part of the algorithm, typically taking more than 90% of the execution time. Both indirect and direct interaction kernels have been implemented on the CPU, based on template recursion and SSE/AVX intrinsics to achieve a high degree of instruction and data level parallelism, respectively. Since the direct and indirect interactions can be computed independently, and since N-body problems are characterized by a high operational intensity, the direct interaction computations are also implemented with a CUDA GPU kernel. The regular structure of bricks allows for a significant reduction in the host-to-device memory transfer. The GPU implementation maps all direct interactions for a destination brick on one streaming multiprocessor (SM), so that each CUDA block has 8^3 threads. We use the shared memory on an SM to load up to 64 source point locations and weights in parallel, and let each CUDA thread in the SM evaluate those sources for its own destination point.

The accuracy of the multipole algorithm is characterized by two user-specified parameters, the order of the expansions p and the opening angle θ. From a performance point of view, the first parameter dictates the cost of creating the multipole expansions and evaluating the indirect interactions, whereas the second parameter controls the number of direct and indirect interactions. Numerically, the combination of these parameters controls the error in the results. In practice, for a known required accuracy, these two parameters allow a degree of freedom that is tuned to maximize overlap between CPU and GPU computation time.

4. RESULTS

We run our performance tests on a single 16-core AMD Interlagos 6272 compute node, with an NVIDIA Tesla K20X GPU, as available on the Cray XK7 system Tödi at the Swiss Supercomputing Center (CSCS). The peak performance of a compute node, using all 8 FPUs, is 268.8 GFLOP/s, and the listed peak performance of the K20X is 3.95 TFLOP/s. Both the CPU and GPU support FMA instructions, so the upper performance bounds of our kernels are adjusted for the FMA-ratio found in their instructions. We set the algorithm parameters to $p = 6$ and $\theta = 0.5$.

We measure performance for the flow past an impulsively started sphere at $Re = 550$ during the first 2500 timesteps (up to non-dimensional time $T = 1.5$) at two different maximum resolution levels. The number of direct interactions per solve varied between 10^3 and 10^5, whereas the number of indirect interactions varied between 10^2 and 10^4. The evaluation of the direct interactions on the CPU achieves a performance of 118.6 GFLOP/s, or 74% of the predicted upper bound and 44% of the peak performance. On the GPU, the direct interactions reach 2.2 TFLOP/s, or 96% and 55% of the upper bound and peak performance, respectively. Performing the direct interactions on the GPU delivers a speed-up of about 17. The indirect interactions achieve a performance of 135.6 GFLOP/s, or 81% of the predicted upper bound and 50% of the peak performance.

5. CONCLUSIONS

We presented a hybrid CPU/GPU multipole-based N-body solver for multiresolution grids that achieves more than 40% of the peak performance of a Cray XK7 compute node. The algorithm will be used for flow optimizations related to bluff-body flows and self-propelled swimmers. Future work will focus on the extension to distributed memory architectures.

Acknowledgments

We thank Dr. Peter Messmer (NVIDIA) for several helpful discussions on improving the performance of the direct interaction evaluations on the GPU.

6. REFERENCES

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algoritm. *Nature*, 324:446–449, 1986.

[2] P. Chatelain and P. Koumoutsakos. A Fourier-based elliptic solver for vortical flows with periodic and unbounded directions. *Journal of Computational Physics*, 229(7):2425 – 2431, 2010.

[3] L. Greengard and V. Rohklin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, Dec 1987.

[4] P. Koumoutsakos. Inviscid axisymmetrization of an elliptical vortex. *Journal of Computational Physics*, 138(2):821–857, 1997.

[5] A. Rahimian, I. Lashuk, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, S. Veerapaneni, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *SC10*, 2010.

[6] A. Robison, M. Voss, and A. Kukanov. Optimization via reflection on work stealing in TBB. In *2008 IEEE International Symposium on Parallel & Distributed Processing, Vols 1-8*, pages 598–605, 2008.

[7] D. Rossinelli, B. Hejazialhosseini, M. Bergdorf, and P. Koumoutsakos. Wavelet-adaptive solvers on multi-core architectures for the simulation of complex systems. *Concurrency and Computation: Practice and Experience*, 23(2):172–186, Feb 2011.

[8] D. Rossinelli, B. Hejazialhosseini, D. Spampinato, and P. Koumoutsakos. Multicore/multi-GPU accelerated simulations of multiphase compressible flows using wavelet adapted grids. *SIAM Journal On Scientific Computing*, 33(2):512–540, 2011.

[9] R. Yokota and L. Barba. Hierarchical N-body simulations with autotuning for heterogeneous systems. *Computing in Science & Engineering*, 14(3):30–39, 2012.